Adversarial Search

Toolbox so far

- Uninformed search
  - BFS, DFS, uniform cost search
- Heuristic search
  - A*

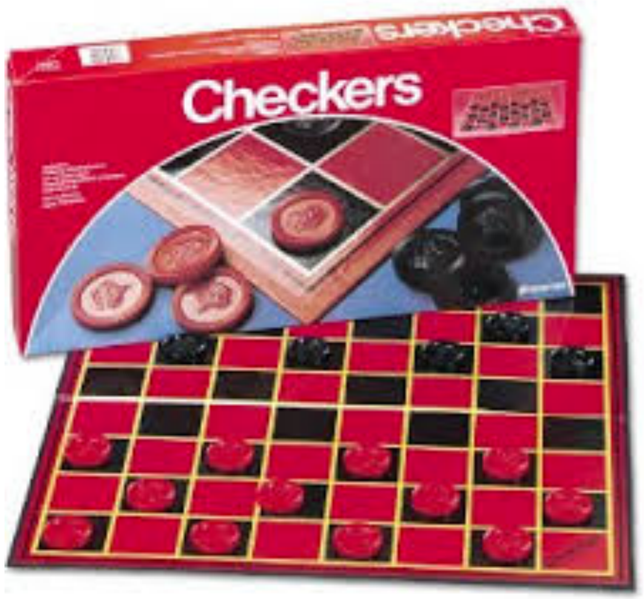
Common environmental factors: static, discrete, fully observable, deterministic actions.
Also: single agent, non-episodic.

Kick it up a notch!

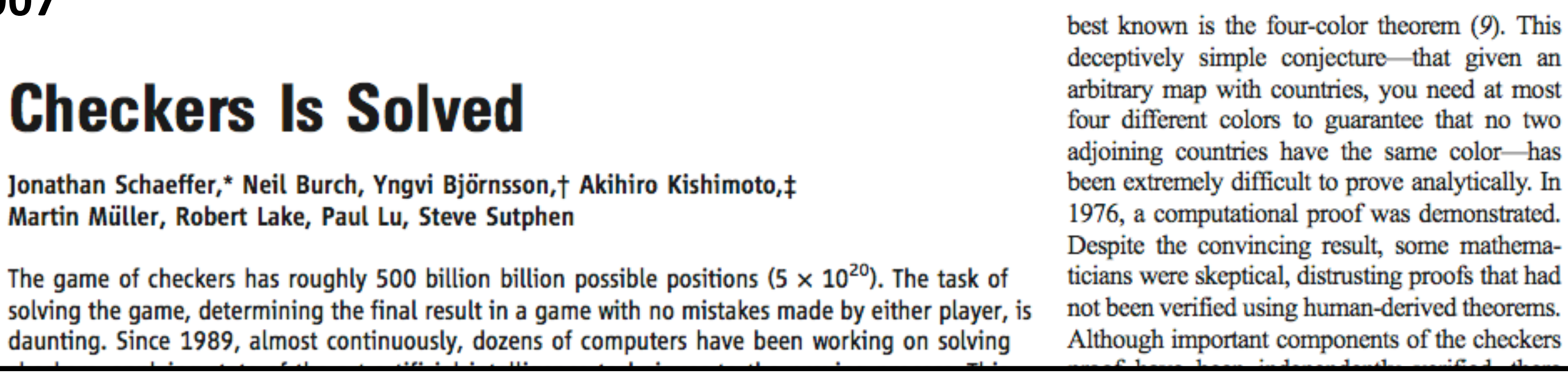
- Add a second agent, but not controlled by us.
- Assume this agent is our adversary.
- Environment (for now)
  - Still static
  - Still discrete
  - Still fully observable (for now)
  - Still deterministic (for now)

Games!

- Deterministic, turn-taking, two-player, zero-sum games of perfect information.



2007

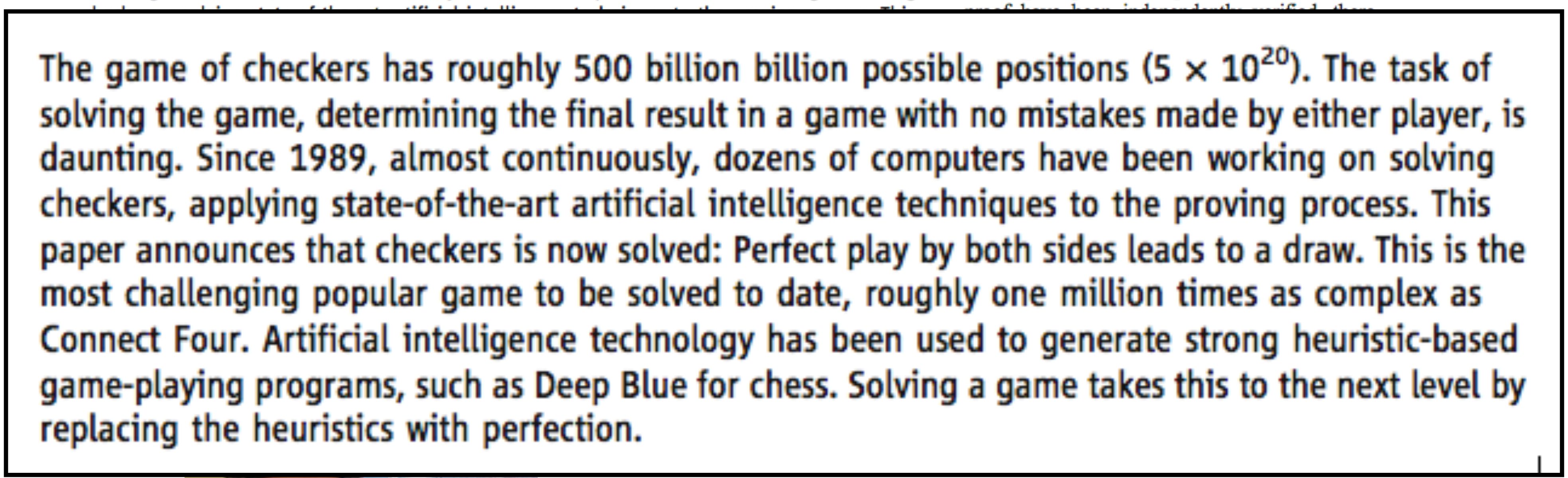


Checkers Is Solved

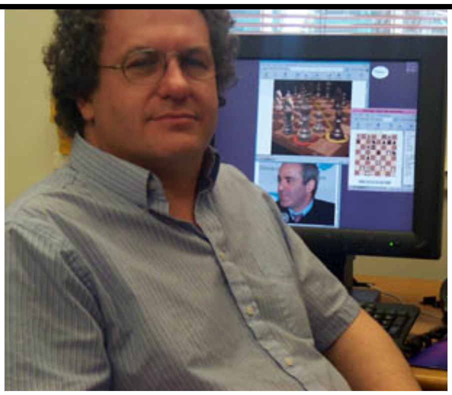
Jonathan Schaeffer,* Neil Burch, Yngvi Björnsson,† Akihiro Kishimoto,‡
Martin Müller, Robert Lake, Paul Lu, Steve Sutphen

The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving

best known is the four-color theorem (*9*). This deceptively simple conjecture—that given an arbitrary map with countries, you need at most four different colors to guarantee that no two adjoining countries have the same color—has been extremely difficult to prove analytically. In 1976, a computational proof was demonstrated. Despite the convincing result, some mathematicians were skeptical, distrusting proofs that had not been verified using human-derived theorems. Although important components of the checkers



The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving checkers, applying state-of-the-art artificial intelligence techniques to the proving process. This paper announces that checkers is now solved: Perfect play by both sides leads to a draw. This is the most challenging popular game to be solved to date, roughly one million times as complex as Connect Four. Artificial intelligence technology has been used to generate strong heuristic-based game-playing programs, such as Deep Blue for chess. Solving a game takes this to the next level by replacing the heuristics with perfection.



UNIVERSITY OF
ALBERTA
EDMONTON, ALBERTA, CANADA

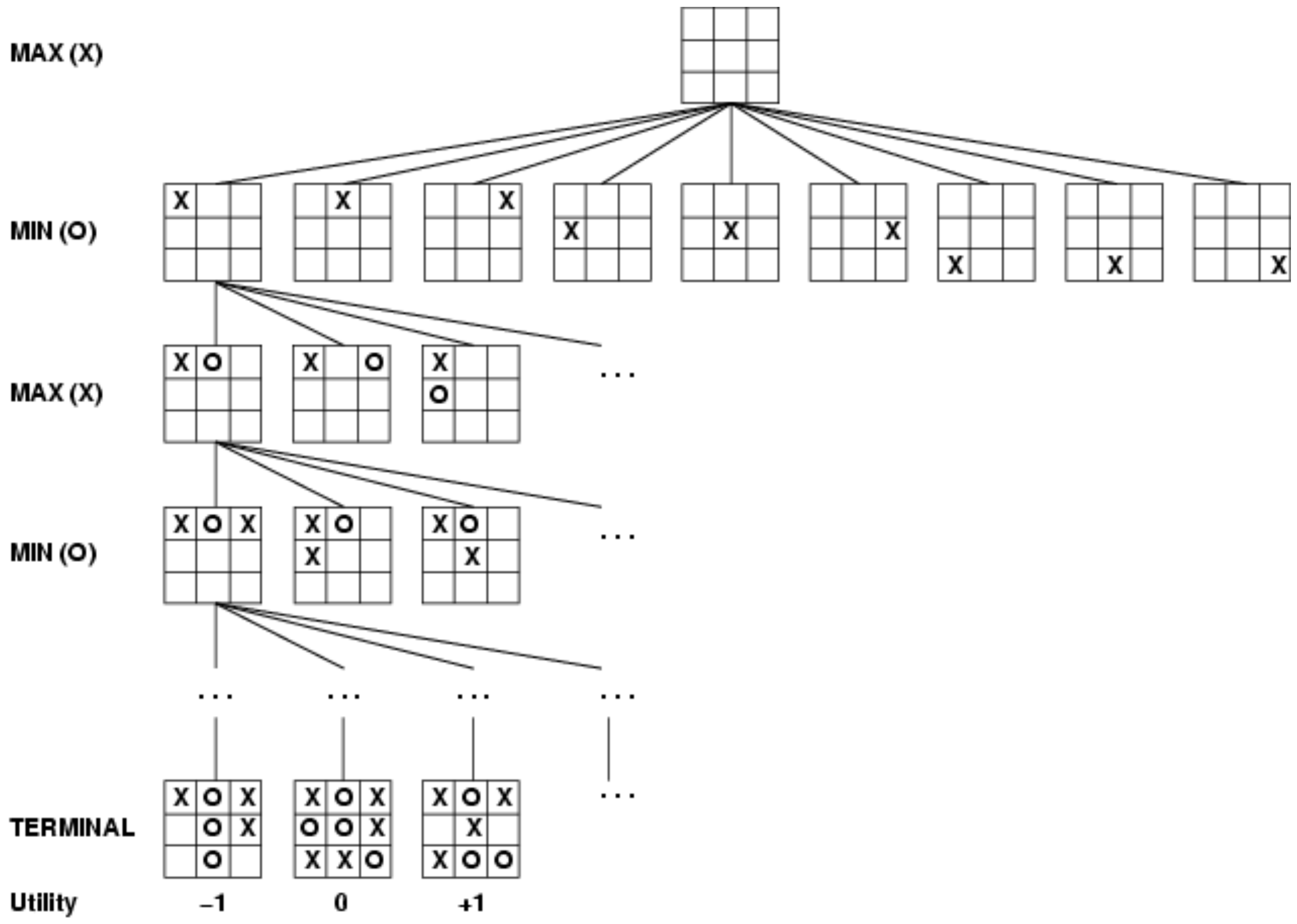
DEPARTMENT OF
COMPUTING SCIENCE

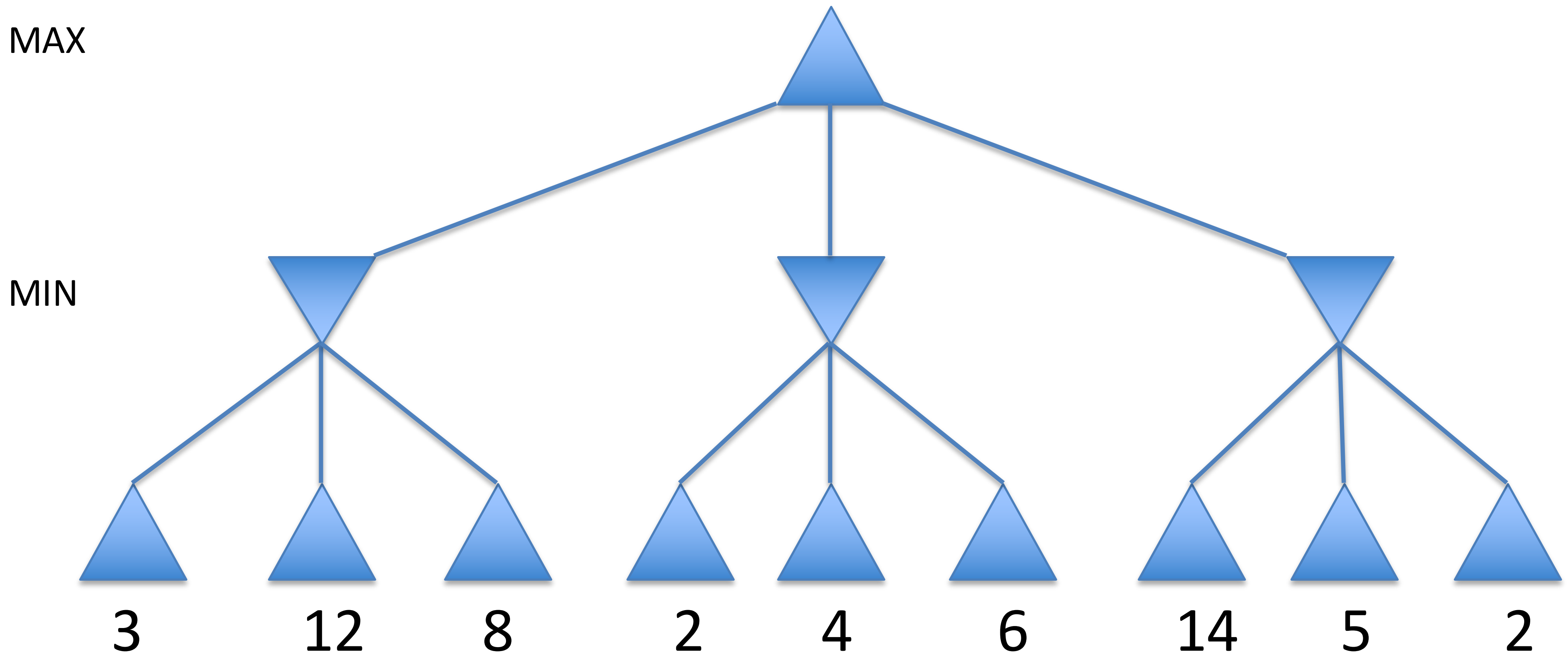
Adversarial search

- Still search!
  - But another agent will alternate actions with us.
- Main new concept:
  - Two players are called MAX and MIN.
  - Only works for zero-sum games.
    - Strictly competitive (no cooperation).
    - What is good for me is equally bad for my opponent (in regards to winning and losing).
  - Most “normal” 2-player games are zero-sum.

- Most all of our concepts from state-space search transfer here.
- Initial state
- PLAYER(s): Defines who makes the next move at a state.
- ACTIONS(s): Returns the set of legal moves in a state.
- RESULT(s, a): Returns what state you go into (transition model)
- TERMINAL-TEST(s): Returns true if s is a terminal state.
- UTILITY(s, p): Numeric value of a terminal state s for player p.

Game Tree




MAX
MIN
3
12
8
2
4
6
14
5
2

Minimax algorithm

- Select the best move for you, assuming your opponent is selecting the best move for themselves.
- Works like DFS.

Minimax algorithm

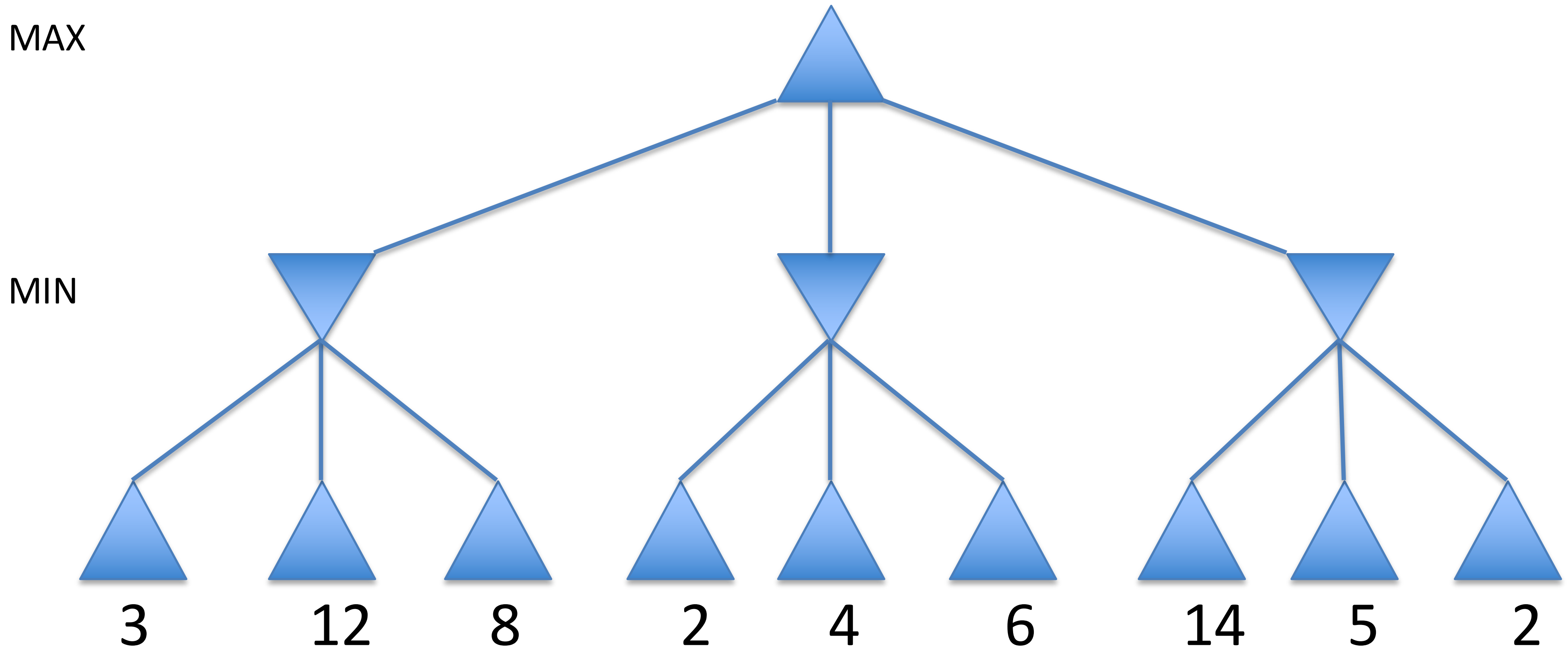
minimax(s) =

utility(s)	if s is terminal
$\max_{a \text{ in actions(s)}}$ minimax(result(s, *a*))	if player(s)=MAX
$\min_{a \text{ in actions(s)}}$ minimax(result(s, *a*))	if player(s)=MIN

result(s, a) means the new state generated by taking action *a* in state *s*.

Properties of minimax

- Complete?
  - Yes (assuming tree is finite)
- Optimal?
  - Yes (assuming opponent is also optimal)
- Time complexity: O(b^m)
- Space complexity: O(bm)   (like DFS)
- But for chess, b ≈ 35, m ≈ 100, so this time is completely infeasible!



$$\text{minimax}(s) = \begin{cases} \text{utility}(s) & \text{if } s \text{ is terminal} \\ \max_{a \text{ in actions}(s)} \text{minimax}(\text{result}(s, a)) & \text{if player}(s)=\text{MAX} \\ \min_{a \text{ in actions}(s)} \text{minimax}(\text{result}(s, a)) & \text{if player}(s)=\text{MIN} \end{cases}$$

- Problem: minimax takes too long.
- Solution: improve algorithm to ignore parts of the tree that will definitely not be used (assuming both players play optimally).

- New algorithm: ***minimax with alpha-beta pruning.***
- Idea: for each node, keep track of the range of possible values that minimax could produce for that node.

Alpha-beta pruning

- Each node in the game tree needs two extra variables, called alpha and beta.
- Alpha and beta are inherited from parent nodes.
  - alpha = highest-value choice we’ve found so far (best move for MAX)
  - beta = lowest-value choice we’ve found so far (best choice for MIN)
- If at a MAX node, we see a child node that has a value >= than beta, **short-circuit**.
- If at a MIN node, we see a child node that has a value <= than alpha, **short-circuit**.

- The results of alpha-beta depend on the order in which moves are considered among the children of a node.
- If possible, consider better moves first!

Real-world use of alpha-beta

- (Regular) minimax is normally run as a preprocessing step to find the optimal move from every possible situation.
- Minimax with alpha-beta can be run as a preprocessing step, but might have to re-run during play if a non-optimal move is chosen.
- Save states somewhere so if we re-encounter them, we don't have to recalculate everything.

Real-world use of alpha-beta

- States get repeated in the game tree because of *transpositions*.
- When you discover a best move in minimax or alpha-beta, save it in a lookup table (probably a hash table).
  - Called a *transposition table*.

Real-world use of alpha-beta

- In the real-world, alpha-beta does not "pre-generate" the game tree.
  - The whole point of alpha-beta is to not have to generate all the nodes.
- The DFS part of minimax/alpha-beta is what generates the tree.